Resource Management System for Cloud Computing Services on Virtual Machines

Mohammad. Akram

Department of Computer Science & Engineering
Christu Jyoti Institute of Technology and Science
Yeswanthapur, Jangaon, Warangal (Dist), Andhra Pradesh, India

Abstract—Cloud computing allows business customers to scale up and down their resource usage based on needs. Many of the touted gains in the cloud model come from resource multiplexing through virtualization technology. In this paper, we present a system that uses virtualization technology to allocate data center resources dynamically based on application demands and support green computing by optimizing the number of servers in use. We introduce the concept of “skewness” to measure the unevenness in the multi-dimensional resource utilization of a server. By minimizing skewness, we can combine different types of workloads nicely and improve the overall utilization of server resources.

We develop a set of heuristics that prevent overload in the system effectively while saving energy used. Trace driven simulation and experiment results demonstrate that our algorithm achieves good performance.

Index Terms—Cloud Computing, Resource Management, Virtualization, Green Computing.

I. Introduction

The elasticity and the lack of upfront capital investment offered by cloud computing is appealing to many businesses. There is a lot of discussion on the benefits and costs of the cloud model and on how to move legacy applications onto the cloud platform. Here we study a different problem: how can a cloud service provider best multiplex its virtual resources onto the physical hardware? This is important because much of the touted gains in the cloud model come from such multiplexing. Studies have found that servers in many existing data centers are often severely under-utilized due to over-provisioning for the peak demand. The cloud model is expected to make such practice unnecessary by offering automatic scale up and down in response to load variation. Besides reducing the hardware cost, it also saves on electricity which contributes to a significant portion of the operational expenses in large data centers.

Virtual machine monitors (VMMs) like Xen provide a mechanism for mapping virtual machines (VMs) to physical resources. This mapping is largely hidden from the cloud users. Users with the Amazon EC2 service, for example, do not know where their VM instances run. It is up to the cloud provider to make sure the underlying physical machines (PMs) have sufficient resources to meet their needs.

VM live migration technology makes it possible to change the mapping between VMs and PMs while applications are running. However, a policy issue remains as how to decide the mapping adaptively so that the resource demands of VMs are met while the number of PMs used is minimized. This is challenging when the resource needs of VMs are heterogeneous due to the diverse set of applications they run and vary with time as the workloads grow and shrink. The capacity of PMs can also be heterogeneous because multiple generations of hardware co-exist in a data center.

Aim to Achieve Two Goals in This Algorithm

Overload Avoidance: the capacity of a PM should be sufficient to satisfy the resource needs of all VMs running on it. Otherwise, the PM is overloaded and can lead to degraded performance of its VMs. green computing: the number of PMs used should be minimized as long as they can still satisfy the needs of all VMs. Idle PMs can be turned off to save energy. There is an inherent trade-off between the two goals in the face of changing resource needs of VMs. For overload avoidance, we should keep the utilization of PMs low to reduce the possibility of overload in case the resource needs of VMs increase later. For green computing, we should keep the utilization of PMs reasonably high to make efficient use of their energy.

In this paper, the design and implementation of an automated resource management system that achieves a good balance between the two goals. We make the following contributions:

A resource allocation system is developed that can avoid overload in the system effectively while minimizing the number of servers used.
We introduce the concept of "skewness" to measure the uneven utilization of a server. By minimizing skewness, we can improve the overall utilization of servers in the face of multi-dimensional resource constraints.

Designing a load prediction algorithm that can capture the future resource usages of applications accurately without looking inside the VMs. The algorithm can capture the rising trend of resource usage patterns and help reduce the placement churn significantly.

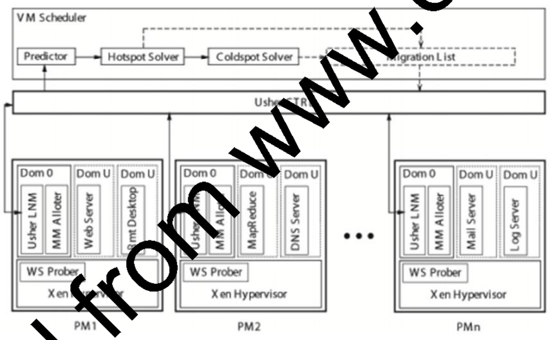

Fig 1: System Architecture

II. Overview

The architecture of the system is presented each PM runs the Xen hypervisor (VMM) which supports a privileged domain 0 and one or more domain U. Each VM in domain U encapsulates one or more applications such as Web server, remote desktop, DNS, Mail, Map/Reduce, etc. We assume all PMs share a backend storage.

The multiplexing of VMs to PMs is managed using the Usher framework. The main logic of our system is implemented as a set of plug-ins to usher. Each node runs an Usher local node manager (LNM) on domain 0 which collects the usage statistics of resources for each VM on that node. The CPU and network usage can be calculated by monitoring the scheduling events in xen. The memory usage with in VM, however, is not visible to the hypervisor. One approach is to infer memory shortage of a VM by observing its swap activities. Unfortunately, the guest OS is required to install a separate swap partition. Furthermore, it may be too late to adjust the memory allocation by the time swapping occurs.

Instead we implemented a working set prober (WS Prober) on each hypervisor to estimate the working set sizes of VMs running on it. We use the random page sampling technique as in the VMware ESX Server.

The statistics collected at each PM are forwarded to the Usher central controller (Usher CTRL) where our VM scheduler runs. The VM Scheduler is invoked periodically and receives from the LNM the resource demand history of VMs, the capacity and the load history of PMs, and the current layout of VMs on PMs.

The scheduler has several components. The predictor predicts the future resource demands of VMs and the future load of PMs based on past statistics. We compute the load of a PM by aggregating the resource usage of its VMs. The details of the load prediction algorithm will be described in the next section. The LNM at each node first attempts to satisfy the new demands locally by adjusting the resource allocation of VMs sharing the same VMM. Xen can change the CPU allocation among the VMs by adjusting their weights in its CPU scheduler. The MM Alloter on domain 0 of each node is responsible for adjusting the local memory allocation.

The hot spot solver in our VM Scheduler detects if the resource utilization of any PM is above the hot threshold (i.e., a hot spot). If so, some VMs running on them will be migrated away to reduce their load. The cold spot solver checks if the average utilization of actively used PMs (APMs) is below the green computing threshold. If so, some of those PMs could potentially be turned off to save energy. It identifies the set of PMs whose utilization is below the cold threshold (i.e., cold spots) and then attempts to migrate away all their VMs. It then compiles a migration list of VMs and passes it to the Usher CTRL for execution.

III. Future Needs for Resources

To predict the future resource needs of VMs. As said earlier, focus is on Internet applications. One solution is to look inside a VM for application level statistics, e.g., by parsing logs of pending requests. Doing so requires modification of the VM which may not always be possible. Instead, we make our prediction based on the past external behaviors of VMs. Our first attempt was to calculate an exponentially weighted moving average (EWMA) using,
TCP-like scheme:

$$E(t) = \alpha * E(t-1) + (1-\alpha) * O(t), 0 \le \alpha \le 1$$

Where $E(t)$ and $O(t)$ are the estimated and the observed load at time t, respectively α reflects a tradeoff between stability and responsiveness.

We use the EWMA formula to predict the CPU load on he DNS server in our university. We measure the load every minute and predict the load in the next minute. Shows the results for $\alpha = 0.7$. Each dot in the figure is an observed value and the curve represents the predicted values. Visually, the curve cuts through the middle of the dots which indicates a fairly accurate prediction. This is also verified by the statistics in Table 1. The parameters in the parenthesis are the α values. W is the length of the measurement window (explained later). The "median" error is calculated as a percentage of the observed value: The "higher" and "lower" error percentages are the percentages of predicted values that are higher or lower than the observed values, respectively. As we can see, the prediction is fairly accurate with roughly equal percentage of higher and lower values.

IV. The Skewness Algorithm

The concept of skewness is introduced to quantify the unevenness in the utilization of multiple resources on a server. Let n be the number of resources we consider and r I be the utilization of the i-th resource. We define the resource skewness of a server p as where r is the average utilization of all resources for server p. In practice, not all types of resources are performance critical and hence we only need to consider bottleneck resources in the above calculation. By minimizing the skewness, we can combine different types of workloads nicely and improve the overall utilization of server resources.

$$skewness(p) = \sqrt{\sum_{i=1}^{n}(\frac{r_i}{\bar{r}} - 1)^2}$$

A. Hot and Cold Spots

Algorithm executes periodically to evaluate the resource allocation status based on the predicted future resource demands of VMs. Define a server as a hot spot if the utilization of and of its resources is above a hot threshold. This indicates that the server is overloaded and hence some VMs running on it should be migrated away. We define the temperature of a hot spot p as the square sum of its resource utilization beyond the hot threshold:

$$temperature(p) = \sum_{r \in R}(r - r_t)^2$$

Where R is the set of overloaded resources in server p and r t is the hot threshold for resource r. (Note that only overloaded resources are considered in the calculation.) The temperature of a hot spot reflects its degree of overload. If a server is not a hot spot, its temperature is zero. Define a server as a cold spot if the utilizations of all its resources are below a cold threshold. This indicates that the server is mostly idle and a potential candidate to turn off to save energy. However, we do so only when the average resource utilization of all actively used servers (i.e., APMs) in the system is below a green computing threshold. A server is actively used if it has at least one VM running. Otherwise, it is inactive. Finally, we define the warm threshold to be a level of resource utilization that is sufficiently high to justify having the server running but not so high as to risk becoming a hot spot in the face of temporary fluctuation of application resource demands.

Different types of resources can have different thresholds. For example, we can define the hot thresholds for CPU and memory resources to be 90% and 80%, respectively. Thus a server is a hot spot if either its CPU usage is above 90% or its memory usage is above 80%.

B. Hot Spot Mitigation

We sort the list of hot spots in the system in descending temperature (i.e., we handle the hottest one first). Our goal is to eliminate all hot spots if possible. Otherwise, keep their temperature as low as possible. For each server p, we first decide which of its VMs should be migrated away. We sort its list of VMs based on the resulting temperature of the server if that VM is migrated away. We aim to migrate away the VM that can reduce the server's temperature the most. In case of ties, we select the VM whose removal can reduce the skewness of the server the most. For each VM in the list, we see if we can find a destination server to accommodate it. The server must not become a hot spot after accepting this VM. Among all such servers, we select one whose skewness can be reduced the most by accepting this VM. Note that this reduction can be negative which means we select the server whose skewness increases the least. If a destination server is found, we record the migration of the VM to that server and update the predicted load of related servers. Otherwise, we move on to the next VM in the list and try to find a destination server for it. As long as we can find a destination server for any of its VMs, we consider this run of the algorithm a success and then move on to the next hot spot. Note that each run of the algorithm migrates away at most one VM from the overloaded server. This does not necessarily eliminate the hot spot, but at least reduces its temperature. If it remains a hot spot in the next decision run, the algorithm will repeat this process. It is possible to design the algorithm so that it can migrate away multiple VMs during each run. But this can add more load on the related servers during a period when they are already overloaded. We decide to use this more conservative approach and leave the system some time to react before initiating additional migrations.

C. Green Computing

When the resource utilization of active servers is too low, some of them can be turned off to save energy. This is handled in our green computing algorithm. The challenge here is to reduce the number of active servers during low load without sacrificing performance either now or in the future. We need to avoid oscillation in the system.

Green computing algorithm is invoked when the average utilizations of all resources on active servers are below the green computing threshold. We sort the list of cold spots in the system based on the ascending order of their memory size. Since we need to migrate away all its VMs before we can shutdown an under-utilized server, we define the memory size of a cold spot as the aggregate memory size of all VMs running on it. Recall that our model assumes all VMs connect to a shared back-end storage. Hence, the cost of a VM live migration is determined mostly by its memory footprint. The complementary file explains why the memory is a good measure in depth. And eliminate the cold spot with the lowest cost first. For a cold spot p, check can migrate all its VMs somewhere else. For each VM on p, we try to find a destination server to accommodate it. The resource utilizations of the server after accepting the VM must be below the warm threshold. While we can save energy by consolidating under-utilized servers, overdoing it may create hot spots in the future. The warm threshold is designed to prevent that. If multiple servers satisfy the above criterion, we prefer one that is not a current cold spot. This is because increasing load on a cold spot reduces the likelihood that it can be eliminated. However, we will accept a cold spot as the destination server if necessary. All things being equal, we select a destination server whose skewness can be reduced the most by accepting this VM. If we can find destination servers for all VMs on a cold spot, we record the sequence of migrations and update the predicted load of related servers. Otherwise, we do not migrate any of its VMs. The list of cold spots is also updated because some of them may no longer be cold due to the proposed VM migrations in the above process.

The above consolidation adds extra load onto the related servers. This is not as serious a problem as in the hot spot mitigation case because green computing is initiated only when the load in the system is low. Nevertheless, we want to bound the extra load due to server consolidation. We restrict the number of cold spots that can be eliminated in each run of the algorithm to be no more than a certain percentage of active servers in the system. This is called the consolidation limit.

Note that we eliminate cold spots in the system only when the average load of all active servers (APMs) is below the green computing threshold. Otherwise, we leave those cold spots there as potential destination machines for future offloading. This is consistent with our philosophy that green computing should be conducted conservatively.

D. Consolidated Movements

The movements generated in each step above are not executed until all steps have finished. The list of movements are then consolidated so that each VM is moved at most once to its final destination. For example, hot spot mitigation may dictate a VM to move from PM A to PM B, while green computing dictates it to move from PM B to PM C. In the actual execution, the VM is moved from A to C directly.

V. Conclusion

We have presented the design, implementation, and evaluation of a resource management system for cloud computing services. Our system multiplexes virtual to physical resources adaptively based on the changing demand. We use the skewness metric to combine VMs with different resource characteristics appropriately so that the capacities of servers are well utilized. Our algorithm achieves both overload avoidance and green computing for systems with multi-resource constraints.

References

1. M. Armbrust *et al.*, "Above the clouds: A berkeley view of cloud computing," University of California, Berkeley, Tech. Rep., Feb 2009.
2. L. Siegele, "Let it rise: A special report on corporate IT," in *The Economist*, Oct. 2008.
3. P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proc. of the ACM Symposium on Operating Systems Principles (SOSP'03)*, Oct. 2003.
4. "Amazon elastic compute cloud (Amazon EC2), http://aws.amazon.com/ec2/."
5. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proc. of the Symposium on Networked Systems Design and Implementation (NSDI'05)*, May 2005.
6. M. Nelson, B.-H. Lim, and G. Hutchins, "Fast transparent migration for virtual machines," in *Proc. of the USENIX Annual Technical Conference*, 2005.
7. M. McNett, D. Gupta, A. Vahdat, and G. M. Voelker, "Usher: An extensible framework for managing clusters of virtual machines," in *Proc. of the Large Installation System Administration Conference (LISA'07)*, Nov. 2007.
8. T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Black-box and gray-box strategies for virtual achine migration," in *Proc. Of the Symposium on Networked Systems Design and Implementation (NSDI'07)*, Apr. 2007.
9. A. Waldspurger, "Memory resource management in VMware ESX server," in *Proc. of the symposiumon Operating systems design and implementation (OSDI'02)*, Aug. 2002.
10. G. Chen, H. Wenbo, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao, "Energy-aware server provisioning and load dispatching for connection-intensive internet services," in *Proc. of the USENIX Symposium on Networked Systems Design and Implementation (NSDI'08)*, Apr. 2008.